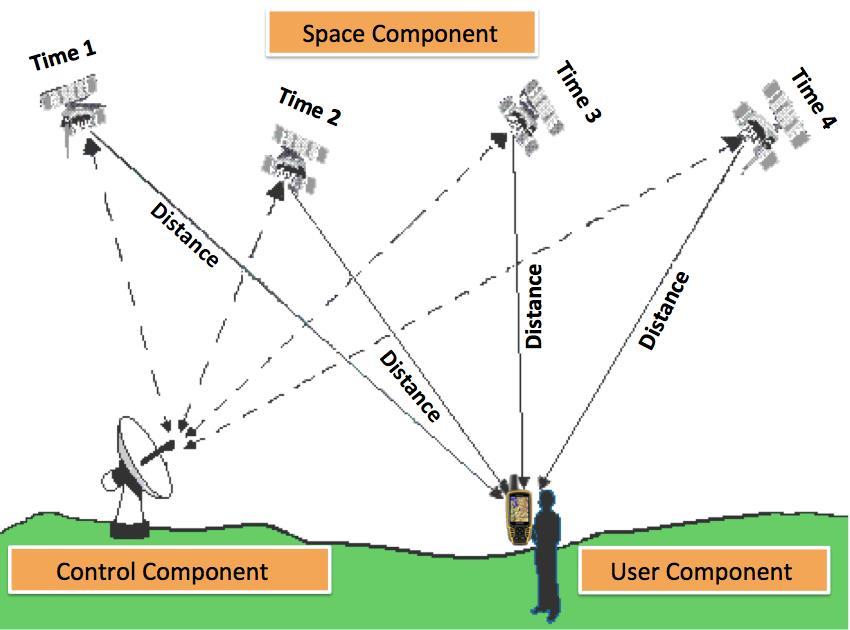
روش کارکرد GPS (Global Positioning System)

GPS چیست؟

GPS یک سیستم راهبری بر پایه ماهواره است که توسط نیروی هوایی ایالات متحده توسعه یافته است، این سیستم اطلاعات موقعیت و زمان برای گیرنده GPS تعیین میکند. گیرنده GPS برای عملکرد خود حداقل نیاز به اتصال به 4 ماهواره دارد، و هر میزان تعداد ماهواره ها بیشتر شود، دقت موقعیت و زمان بیشتر میشود (دلیل افزایش دقت در ادامه واضح تر میشود).



روش کار GPS

ماهواره های GPS به صورتی در مدارهای زمین قرار گرفته اند که در هر زمانی حداقل 4 عدد از آن ها برای گیرنده در سطح زمین قابل دید است، هر یک از این ماهواره ها اطلاعات زمان و موقعیت فعلی خود را برای گیرنده ارسال میکنند. این اطلاعات به وضوح توسط سیگنال هایی که با سرعت نور در مسافت را طی میکنند ارسال میشود، گیرنده با توجه به زمانی که سیگنال را دریافت و اخلاف آن با زمانی که اطلاعات داخل پیام نوشته شده است، میزان مسافت خود را با ماهواره بدست میاورد.

با بدست آوردن فاصله خود از 3 ماهواره میتوانیم طول و عرض جغرافیای موقعیت خود را متوجه شویم، اگر فاصله را از 4 ماهواره داشته باشیم، ارتفاع جغرافیایی نیز بدست میاید.

چالش های مهندسی که طراحی GPS با آن ها مواجه شده

همان طور که از روش کارکرد GPS متوجه شدیم، چندین پارامتر مهم در محاسبه موقعیت نقش دارد، و این عوامل مشکلات زیر را ایجاد میکنند

1. هماهنگ سازی زمانی بین ماهواره ها و گیرنده
2. موقعیت دقیق ماهواره در فضا
3. محاسبه حساس تاخیر زمانی بین ماهواره و گیرنده
4. تداخل رادیوی با دیگر فرکانس ها

راه حل های فعلی

هماهنگ سازی زمانی

برای هماهنگ سازی زمانی بین ماهواره ها، در داخل ماهواره ها از ساعت های اتمی (GPSDO) استفاده شده است تا زمان را دقیق و به روز حفظ کند. هماهنگ سازی زمانی در گیرنده نیز به این صورت است که تفاوت زمانی توسط گیرنده با استفاده از ماهواره چهارم انجام میشود، بحث در مورد جزئیات بیشتر و معادلات ریاضی از بحث ما خارج است.



موقعیت دقیق ماهواره در فضا

این مشکل با استفاده از نظارت پایگاه های زمینی که در سطح زمین پراکنده شده‌اند حل شده است، به این صورت که پایگاه های زمینی انجام تمام اندازه گیری ها، موقعیت دقیق هر ماهواره را ارسال میکنند.

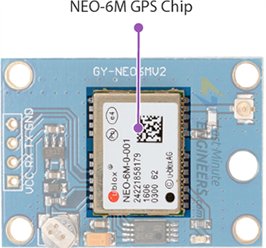
منبع:

<https://www.geeksforgeeks.org/how-gps-works/>

بررسی سخت افزار (ماژول)

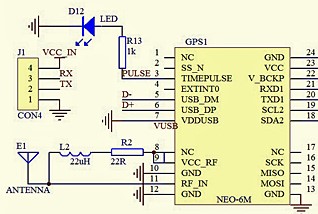
مقدمه

ماژولی که در این پروژه استفاده خواهد شد، ماژول نسبتا معروف NEO-6M از شرکت ublox است، این ماژول قابلیت اتصال همزمان به 22 ماهواره در 50 کانال را دارد، و میتواند به صورت بالقوه دقت بسیار بالایی ارائه دهد. از قابلیت های دیگر آن حساسیت بالا -161dB و مصرف پایین 45mA در حالت فعال.



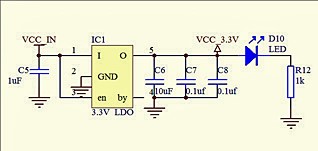
Indicator

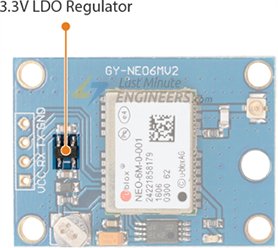
اگر به شماتیک چیپ توجه کنیم، یکی از پایه های آن با نام TimePulse مشخص شده است، وظیفه این پایه به این صورت میباشد که اگر گیرنده ما در اتصال به ماهواره ها موفق باشد، شروع به تولید پالس میکند، به این صورت اگه این پایه به یک LED متصل شود، LED شروع به چشمک زدن میکند، در غیر این صورت به صورت High خواهد ماند. (در ادامه مطلب شماتیک را کامل خواهیم کرد)



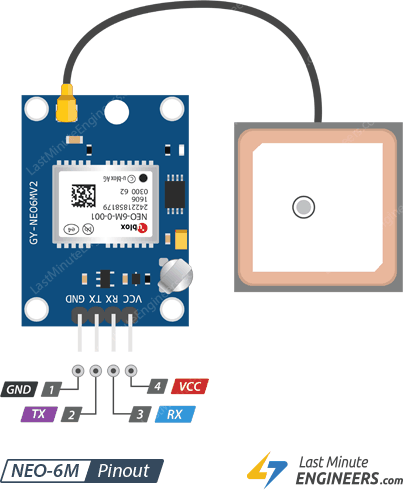
LDO

ولتاژ کاری NEO-6M در محدوده 2.7v – 3.6 است، به همین دلیل اگر به شماتیک توجه کنیم، طراح با استفاده از یک LDO (Low Dropout Regulator) 3.3v در این حالت خاص MIC5205 امکانی را فراهم کرده است، تا بتوانیم ماژول را مستقیما به تغذیه 5 ولت متصل نماییم.





Pinout



VCC: connect to 5v

GND: connect to ground

TX: (UART) connect to RX STM32

RX: (UART) connect to TX STM32

\*\*\* در واقع پایه RX برای فرستادم دستور به ماژول میتواند مورد استفاده قرار گیرد، اما کاربر آن برای ما فقط فرستادن اولین بایت برای ماژول به دلیل ست کردن baud rate است. \*\*\*

استاندارد NMEA

مقدمه

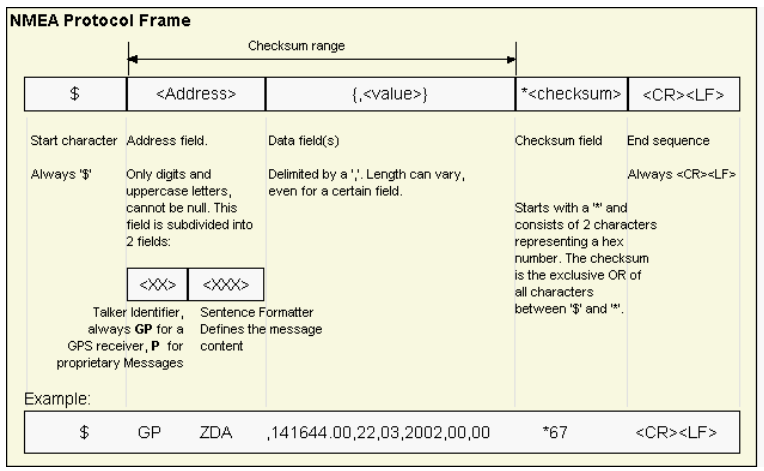
خروجی ماژول با استاندارد (یا فرمت) NMEA (National Marine Electronics Association)، این اطلاعات توسط ماژول به خروجی خود (یعنی پایه TX در این مورد) ارسال میشود.

تعداد دفعاتی که این دیتا ابدیت میشود و فرستاده میشود به نرخ ابدیت ماژول مرتبط است که در حالت پیش فرض روی 1Hz میباشد و تا 5Hz نیز میتوان افزایش داد.

شکل کلی

در شکل زیر فرمت کلی استاندارد آورده شده است، میتوانیم به صورت زیر استاندارد را خلاصه کنیم

Source: <https://www.nmea.org/>



$GP(define message meaning – 3 chars), (message)\*(checksum – 2 chars)

message شامل یک سری اعداد و حروف میباشد که با استفاده از ‘,’ از یکدیگر جدا شده اند، این پیام به خودی خود معنایی ندارد، و معنای آن ها با استفاده از define message meaning مشخص میشود که خود شامل 3 کاراکتر است.

در آخر پیام یک checksum وجود دارد که بعد از \* ظاهر میشود که 2 کاراکتر به هگز میباشد و از XOR کردن تمام کاراکتر ها بین $, \* میباشد، در واقع نحوه عملکرد آن را میتوانیم با کد پایتون زیر شبیه سازی کنیم.

message = ""

check\_sum = 0

for char in message:

    check\_sum ^= ord(char)

print(hex(check\_sum))

مثال

$GPVTG,77.52,T,,M,0.004,N,0.008,K,A\*06

$GP(VTG :meaning),(77.52,T,,M,0.004,N,0.008,K,A :message)\*(06 :checksum)

در مثال بالا نحوه عمکرد خروجی به صورت کلی واضح است، در آینده به بررسی ساختارهای مهم و استخراج اطلاعات از آن ها خواهیم پرداخت.

برای چک کردن checksum هم میتوانیم به صورت زیر عمل کنیم

message = "GPVTG,77.52,T,,M,0.004,N,0.008,K,A"

check\_sum = 0

for char in message:

    check\_sum ^= ord(char)

print(hex(check\_sum))   #output 0x6

فرمت طول و عرض

در فرمت NMEA طول و عرض جغرافیایی در واقع در فرمت ترکیبی درجه و زمان برگشت داده میشود و برای بدست آوردن طول و عرض مجبور به تبدیل آن ها به صورت زیر هستیم

\* فرمت زمان میتواند در پیام های مختلف معنی دقیقه یا ثانیه و یا ترکیبی داشته باشد، که در ادامه در بررسی ساختار هر پیام بررسی خواهد شد.

مثال

Latitude of **4717.112671** North and Longitude of **00833.914843** East

*ساختمان های مهم در خروجی*

RMC

Location (2D) + Time + Speed + Date

$GPRMC, 123519.00, A, 4807.038, N, 01131.000, E,022.4, 084.4, 230394, 003.1, W\*6A

|  |  |
| --- | --- |
| RMC | Recommended Minimum Coordinates |
| 123519.00 | Current Time in UTC – 12:35:19.00 |
| A | A=Active |
| 4807.038, N | Latitude 48 deg, 07.038’ N |
| 01131.000, E | Longitude 11 deg 31.000’N |
| 022.4 | Speed over the ground in knots |
| 084.4 | Track angle in degrees True |
| 230318 | Date – 23rd of March 2018 |
| 003.1, W | Magnetic Variation |
| \*6A | Checksum |

تبدیل knots به kms

GGA

Location (3D) + Time + Number of Satellite

$GPGGA, 123519.00, 4807.038, N, 01131.000, E, 1, 08, 0.9, 545.4, M, 46.9, M, , \*47

|  |  |
| --- | --- |
| GGA | Global Positioning System Fix Data |
| 123519.00 | Current time in UTC – 12:35:19.00 |
| 4807.038, N | Latitude 48 deg 07.038′ N |
| 01131.000, E | Longitude 11 deg 31.000′ E |
| 1 | GPS fix |
| 08 | Number of satellites being tracked |
| 0.9 | Horizontal dilution of position |
| 545.4, M | Altitude in Meters (above mean sea level) |
| 46.9, M | Height of geoid (mean sea level) |
| (Empty field) | Time in seconds since last DGPS update |
| (Empty field) | DGPS station ID number |
| \*47 | Checksum |

|  |  |
| --- | --- |
| GPS fix |  |
| 0 | No Fix |
| 1 | Standard GPS |
| 2 | Differential GPS |
| 6 | Estimated GPS |

\*به صورت کلی GPS Fix ارزش زیادی ندارد و در برنامه نویسی از آن استفاده نخواهیم کرد.

انواع دیگر

تمامی اطلاعات مفید از خروجی های قبلی بدست میاید، انواع دیگر خروجی ها نیز وجود دارد که یا اطلاعات کمی در اختیار ما قرار میدهند، و یا اطلاعات تخصصی دارند که برای ما کاربردی ندارد.

GLL (Location 2D + Time)

GSA (Satellite Number)

GSV (Satellite Information like SNR)

VTG (Speed)

نکته بسیار مهم

دقت پارامترهای برگشت داده شده(تعداد اعشار)، به کیفیت آنتن و تعداد ماهواره ها وابسته است، و گاه ممکن است ماژول ما تا دقایقی نتواند بعضی از داده ها را بدست آورد. اگر پارامتری وجود نداشته باشد، در دیتای برگشت داده شده، جای آن خالی خواهد بود. این موارد باید در برنامه نویسی مورد توجه قرار گیرد.

برای مثال ممکن است زمان بدون اعشار برگشت داده شود، و طول و عرض جغرافیایی بین دو تا 5 رقم اعشار متفاوت است. (این دقت در دیتاشیت آورده نشده است و تجربی است)

\*\*\* در حالت عملی با آنتن فعال متصل شده به ماژول و اتصال به 5 ماهواره، دقت تا 5 رقم اعشار بدست آمد.

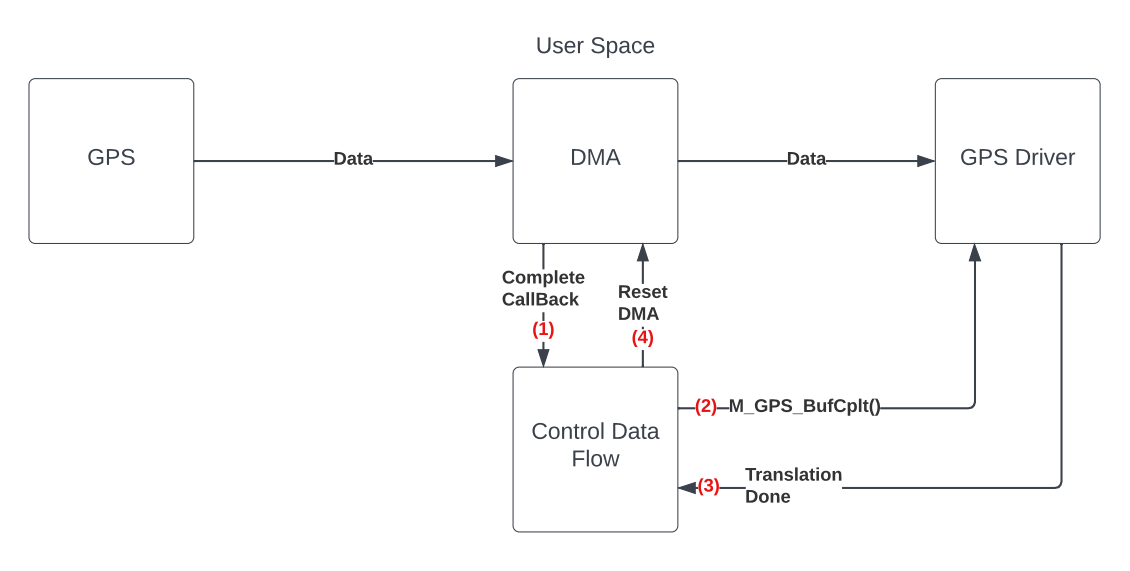
مثال از دقت

|  |  |
| --- | --- |
| دقت طول و عرض جغرافیایی (تعداد اعشار) | میزان خطا در واحد متر |
| 4 | 11.1 m |
| 5 | 1.11 m |
| 6 | 0.111 m |
| 7 | 1.11 cm |

برنامه نویسی

چارت کلی

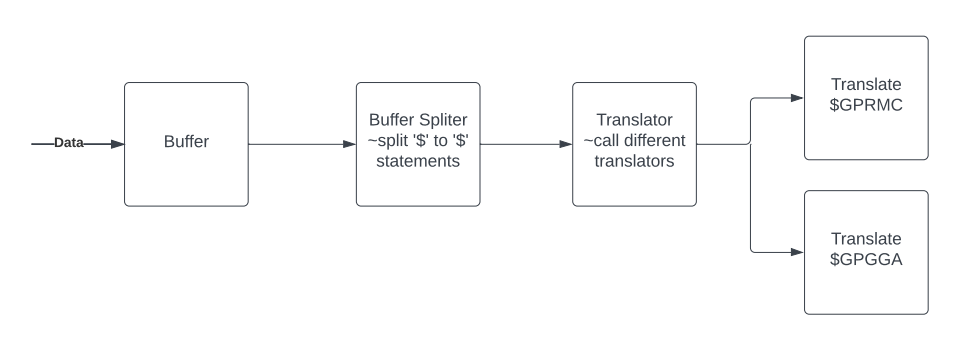
به صورت از دید بالا میتوانیم data flow از ماژول GPS تا درایور را میتوانیم به صورت زیر طراحی کنیم



نگاه به آینده

برنامه ریزی ماژول با استفاده از دستور PUBX و کامل کردن کتابخانه.

طراحی داخلی درایور را میتوانیم به صورت زیر طراحی کنیم



ساختار فایل ها در درایور

در طراحی تمامی درایورها مانند کتابخانه ها فایل به صورت head, core, tests داریم

نام گذاری ها را نیز به صورت زیر انجام میدهیم

M-GPS.h

M-GPS.c

M-GPS-Test.c

متغیر ها و ثبات استفاده شده در درایور

/\* defines ------------------------------------------------------------------\*/

#define \_\_M\_GPS\_BufLen 300

برای تعیین کردن طول بافر ورودی استفاده شده است.

#define \_\_M\_GPS\_TranslatorBufLen 100

برای تعیین طول بافر مترجم مورد استفاده است، این بافر برای تقسیم بندی جملات و ارسال آن ها به مترجم مورد استفاده است.

#define \_\_M\_GPS\_Knot2Km 1.85184

/\* variables ------------------------------------------------------------------\*/

char \_\_M\_GPS\_Buf[\_\_M\_GPS\_BufLen];

char \_\_M\_GPS\_TranslatorBuf[\_\_M\_GPS\_TranslatorBufLen];

uint8\_t \_\_M\_GPS\_localHur;

uint8\_t \_\_M\_GPS\_localMin;

برای ذخیره کردن زمان محلی استفاده میشود، تا بتوانیم زمان محلی را بدست آوریم، برای مثال برای ایران به صورت +3:30 میباشد.

طراحی ساختار داده در درایور

ساختمان خود را با توجه به اطلاعاتی را که در هدف داریم از داده های GPS خود استخراج کنیم، طراحی میکنیم، به این ترتیب به صورت زیر داریم

\* نیاز به تاکید نمیباشد ساختمان خود را در هدر قرار میدهیم

/\* structures -------------------------------------------------------------\*/

struct \_\_M\_GPS\_Time{

    uint8\_t hur;    // hour

    uint8\_t min;    // minute

    uint8\_t sec;    // second

    uint8\_t msc;    // millisecond

};

struct \_\_M\_GPS\_Date{

    uint8\_t yar;    // year

    uint8\_t mon;    // month

    uint8\_t day;    // day

};

struct {

    double lat;     // latitude

    double lon;     // longitude

    double hei;     // height

    double hgo;     // height of geoid

    uint8\_t NoS;    // number of satellites

    uint16\_t spd;   // speed

    struct \_\_M\_GPS\_Time time;

    struct \_\_M\_GPS\_Date date;

} M\_GPS;

نکاتی در حول نامگذاری

1. در ابتدای هر نامگذاری از فرمت M\_GPS\_\* استفاده میکنیم، به این دلیل که درایور نباید با هیچ یک از متغیرهای کاربر یا درایورهای دیگر تداخل داشته باشد، برای مثال اگر به جای M\_GPS\_Buf از نام مرسوم Buf استفاده کنیم، ممکن است با بافرهای تعریف شده توسط کاربر یا درایورهای دیگر به تداخل بخوریم.
2. برای استفاده از داده ها و یا توابع private از “\_\_” در ابتدای نامگذاری استفاده میکنیم، برای مثال

void \_\_M\_GPS\_updateTime(double);

توابع مدیریت درایور

void M\_GPS\_init(UART\_HandleTypeDef uart);

void M\_GPS\_bufInit(UART\_HandleTypeDef uart);

void M\_GPS\_bufCplt(void);

init

یکی از مرسوم ترین توابعی که در ابتدای استفاده از کتابخانه ها مورد استفاده قرار میگیرد میباشد، به این صورت که معمولا با ست کردن پارامترهایی از کتابخانه آن را قابل استفاده میکند.

در این مورد وظیفه آن برای ما، صفر کردن تمامی متغیرهای ساختمان اصلی و ارسال یک بایت داده به ماژول GPS است، در اصل وظیفه این بایت ارسال شده آن است که سرعت uart را برای ماژول GPS مشخص کند.

در مورد توابع updateDate, updateTime صحبت خواهیم کرد، اما با توجه به اسم گذاری واضح است که، دیتای مربوط به زمان و تاریخ در ساختمان اصلی را با توجه به ورودی به روز خواهد کرد.

/\*\*

 \* @breif setting parameters to zero

 \*

 \* @param uart get uart port that GPS is connected on

 \*

 \* @retval None

 \*/

void M\_GPS\_init(UART\_HandleTypeDef uart) {

    // set everything to zero

    M\_GPS.lat = 0;

    M\_GPS.lon = 0;

    M\_GPS.hei = 0;

    M\_GPS.spd = 0;

    \_\_M\_GPS\_updateDate(0);

    \_\_M\_GPS\_updateTime(0);

    M\_GPS\_setLocalTime(0, 0);

    // send data to GPS module to set communication speed

    uint8\_t \_ = '$';

    HAL\_UART\_Transmit(&uart, &\_, 1, 300);

}

bufInit

وظیفه این تابع بسیار ساده است، DMA را برای دریافت داده ها تنظیم میکند.

/\*\*

 \* @breif initialize GPS buffer to save incoming data

 \*

 \* @param None

 \*

 \* @retval None

 \*/

void M\_GPS\_bufInit(UART\_HandleTypeDef uart){

    memset(\_\_M\_GPS\_Buf, 0, \_\_M\_GPS\_BufLen);

دیتای 0 را به تعداد \_\_M\_GPS\_BufLen خانه در \_\_M\_GPS\_Buf کپی میکند، به زبانی ساده، بافر را به صورت کلی صفر میکند

    HAL\_UART\_Receive\_DMA(&uart, (uint8\_t\*) \_\_M\_GPS\_Buf, \_\_M\_GPS\_BufLen);

}

bufCplt

زمانی که بافر توسط DMA کاربر پر میشود، وقفه مرتبط با آن صدا زده میشود، کاربر میتواند در آن لحظه با فراخوانی این تابع اطلاعات را از بافر استخراج نماید.

روش عملکرد این تابه بخ تیم صورت است که، جمله بندی های $ تا $ را جداسازی میکند و سپس سعی میکند با فراخوانی تابع مترجم، آن جمله را ترجمه نماید.

/\*\*

 \* @breif split sentences from buffer from $ to $ and replace blank parameters with space

 \* and send sentences to translator

 \* \*\*\* This function should call in "HAL\_UART\_RxCpltCallback" interrupt

 \*

 \* @param None

 \*

 \* @retval None

 \*/

void M\_GPS\_bufCplt(void) {

    uint16\_t ptr = 0;

    for (uint16\_t i = 0; i < \_\_M\_GPS\_BufLen; i++) {

        if (\_\_M\_GPS\_Buf[i] == '$') {

            \_\_M\_GPS\_translator();

            ptr = 0;

        }

        \_\_M\_GPS\_TranslatorBuf[ptr++] = \_\_M\_GPS\_Buf[i];

        if (\_\_M\_GPS\_Buf[i] == ',')

            \_\_M\_GPS\_TranslatorBuf[ptr++] = ' ';

اگر دو ‘,’ پشت سر هم قرار گرفته باشند، به این معنی است که دیتای مربوط به آن قسمت وجود ندارد، به این ترتیب بین آن دو ‘,’ یک space قرار میدهیم تا مترجم ما متوجه شود این داده موجود نیست.

    }

}

توابع مترجم درایور

void \_\_M\_GPS\_translator(void);

void \_\_M\_GPS\_translateGPRMC(void);

void \_\_M\_GPS\_translateGPGGA(void);

translator

این تابع وظیفه بسیار ساده ای دارد، به صورت که با فراخوانی مترجم های مختلف، سعی میکند جمله موجود در translatorBuf را ترجمه کند.

/\*\*

 \* @breif try to translate by different translator, after success translating

 \* all needed parameters restart GPS buffer to incoming data

 \*

 \* @param None

 \*

 \* @retval None

 \*/

void \_\_M\_GPS\_translator(void) {

    \_\_M\_GPS\_translateGPRMC();

    \_\_M\_GPS\_translateGPGGA();

}

TranslateGPRMC

این تابع جملاتی را که با $GPRMC شروع میشود را ترجمه میکند و اگر بتواند هر گونه دیتایی با توجه به فرمتی که قبلا مورد بحث قرار گرفت استخراج نماید، آن ها را در ساختمان داده اصلی ذخیره خواهد کرد.

\* نکته ای که در اینجا قابل توجه است، این میباشد که %f که اعداد اعشاری رو میتواند بخواند در میکرو کنترل ها پشتیبانی نمیشود، پس دیتا باید ابتدا به صورت string استخراج شود، سپس به عدد تبدیل شود.

/\*\*

 \* @breif obtain parameters from $GPRMC

 \*

 \* @param None

 \*

 \* @retval None

 \*/

void \_\_M\_GPS\_translateGPRMC(void) {

    char time[20] = "";

    char lat[20] = "";  // latitude

    char lon[20] = "";  // longitude

    char speed[10] = "";

    char date[10] = "";

    sscanf(\_\_M\_GPS\_TranslatorBuf,

            "$GPRMC,"

            "%[0-9|.| ], A,"

            "%[0-9|.| ], N,"

            "%[0-9|.| ], E,"

            "%[0-9|.| ],"

            "%\*[0-9|.| ],"

            "%[0-9|.| ]",

            time, lat, lon, speed, date);

    double \_ = atof(time);

    if (\_ != 0)

        \_\_M\_GPS\_updateTime(\_);

در اصل \_ نام متغیر ما در اینجا میباشد، و چون یک داده موقت میباشد و با داده های با معانی متفاوت پر میشود، برای آن این نامگذاری را انجام داده ایم.

    \_ = atof(lat);

    if (\_ != 0)

        \_\_M\_GPS\_updateLocation(\_, &M\_GPS.lat);

    \_ = atof(lon);

    if (\_ != 0)

        \_\_M\_GPS\_updateLocation(\_, &M\_GPS.lon);

    \_ = atof(speed);

    if (\_ != 0)

        M\_GPS.spd = (uint16\_t)(\_ \* \_\_M\_GPS\_Knot2Km);

    \_ = atoi(date);

    if (\_ != 0)

        \_\_M\_GPS\_updateDate((uint16\_t)\_);

}

translateGPGGA

این تابع نیز دقیقا مانند همنوع خود طراحی شده است

/\*\*

 \* @breif obtain parameters from $GPGGA

 \*

 \* @param None

 \*

 \* @retval None

 \*/

void \_\_M\_GPS\_translateGPGGA(void) {

    char number\_of\_satelite[5] = "";

    char height[10] = "";

    char height\_of\_geoid[10] = "";

    sscanf(\_\_M\_GPS\_TranslatorBuf,

                "$GPRMC,"

                "%\*[0-9|.| ],"

                "%\*[0-9|.| ], N,"

                "%\*[0-9|.| ], E, 1,"

                "%[0-9|.| ],"

                "%\*[0-9|.| ],"

                "%[0-9|.| ], M,"

                "%[0-9|.| ], M,",

                number\_of\_satelite, height, height\_of\_geoid);

    double \_ = atoi(number\_of\_satelite);

    if (\_ != 0)

        M\_GPS.NoS = (uint8\_t)\_;

    \_ = atof(height);

    if(\_ != 0)

        M\_GPS.hei = \_;

    \_ = atof(height\_of\_geoid);

    if(\_ != 0)

        M\_GPS.hgo = \_;

}

توابع به روز رسانی دیتا درایور

void \_\_M\_GPS\_updateTime(double);

void \_\_M\_GPS\_updateLocation(double, double \*);

void \_\_M\_GPS\_updateDate(uint32\_t);

updateTime

این تابع برای به روز رسانی زمان در ساختمان اصلی مورد استفاده است، فرمت ورودی به صورت hhmmss.ss میباشد.

/\*\*

 \* @brief update saved time in M\_GPS structure base on newTime

 \*

 \* @param newTime in "hhmmss.ss" format

 \*

 \* @retval None

 \*/

void \_\_M\_GPS\_updateTime(double newTime) {

    double dec\_time = newTime - floor(newTime);

    int int\_time = (int) (newTime);

قسمت اعشاری و صحیح ورودی را از هم جدا میکنیم

    M\_GPS.time.msc = (uint8\_t) (dec\_time \* 100);

قسمت اعشاری معرف milli seconds است و با دو رقم صحیح آن را ذخیره میکنیم

    M\_GPS.time.hur = (int\_time / 10000);

    int\_time %= 10000;

دو رقم ابتدایی معرف ساعت میباشد

    M\_GPS.time.min = (uint8\_t) (int\_time / 100);

    int\_time %= 100;

دو رقم بعدی معرف دقیقه میباشد.

    M\_GPS.time.sec = (uint8\_t) (int\_time);

به این ترتیب دو رقم باقیمانده معرف ثانیه میشود

}

updateLocation

خروجی لوکیشن ماژول که به صورت ترکیبی از درجه و زمان با فرمت dddmm.m\* میباشد را تبدیل به درجه کرده و در داده مرتبط داده شده ذخیره میکند.

/\*\*

 \* @breif convert hybrid location to geographical degree and save in given location

 \*

 \* @param newLocation in dddmm.m\* format

 \* @param location

 \*

 \* @retval None

 \*/

void \_\_M\_GPS\_updateLocation(double newLocation, double \*location) {

    int int\_loc = (int) newLocation;

    double deg = (int)(int\_loc / 100.0);

    double min = newLocation - deg \* 100.0;

دو رقم سمت راست و اعشار معرف زمان به دقیقه میباشد، بقیه آن نیز معرف درجه است، به این ترتیب با جدا کردن این دو قسمت از هم، و سپس تبدیل دقیقه به درجه (همان ساعت)، درجه جغرافیای ما بدست میاید.

    \*location = deg + min / 60.0;

}

updateDate

خروجی تاریخ GPS که به صورت داده یک عدد با فرمت ddmmyy میباشد را day, month, year تبدیل کرده و در ساختمان داده اصلی ذخیره میکند.

/\*\*

 \* @breif update saved date in M\_GPS structure base on newDate

 \*

 \* @param newDate in "ddmmyy" format

 \*

 \* @retval None

 \*/

void \_\_M\_GPS\_updateDate(uint32\_t newDate) {

    M\_GPS.date.day = (uint8\_t)(newDate / 10000);

    newDate %= 10000;

دو رقم سمت چپ (از شش رقم) معرف روز است.

    M\_GPS.date.mon = (uint8\_t)(newDate / 100);

    newDate %= 100;

دو رقم بعدی معرف ماه میباشد

    M\_GPS.date.yar = (uint8\_t)(newDate);

دو رقم باقی مانده نیز معرف سال است

}

توابع دیگر

جدای از دسترسی مستقیمی که کاربر به ساختمان داده دارد و میتواند از آن دیتا های مورد نظر خود را برداشت کند، راه حل user friendly این میباشد که برای بدست آوردن هر داده یک تابع تعریف کنیم، که کاربر بتواند با استفاده از فراخوانی آن به دیتای مورد نظر دسترسی پیدا کند، این توابع به شرح زیر است

// get locations

double M\_GPS\_getLatitude(void);

double M\_GPS\_getLongitude(void);

double M\_GPS\_getHeight(void);

double M\_GPS\_getGeoHeight(void);

// get more information

uint16\_t M\_GPS\_getSpeed(void);

uint8\_t M\_GPS\_getNumberOfSatellites(void);

// get time

void M\_GPS\_getTimeString(char \*str);

uint8\_t M\_GPS\_getHour(void);

uint8\_t M\_GPS\_getMinute(void);

uint8\_t M\_GPS\_getSecond(void);

uint8\_t M\_GPS\_getMilliSecond(void);

// get date

void M\_GPS\_getDateString(char \*str);

uint8\_t M\_GPS\_getDay(void);

uint8\_t M\_GPS\_getMon(void);

uint8\_t M\_GPS\_getYear(void);

تست توابع درایور

دید کلی

برای اطمینان از صحیح بودن توابع که نوشته شده اند، باید برای آن ها unit test بنویسیم که در صورت وجود خطا در آن ها،